Mechanically Verifying the Correctness of an Offline Partial Evaluator

John Hatcliff *

DIKU, Computer Science Department, Copenhagen University **

Abstract. We show that using deductive systems to specify an offline partial evaluator allows one to specify, prototype, and mechanically verify correctness via meta-programming — all within a single framework.

For a λ-mix-style partial evaluator, we specify binding-time constraints using a natural-deduction logic, and the associated program specializer using natural (aka “deductive”) semantics. These deductive systems can be directly encoded in the Elf programming language — a logic programming language based on the LF logical framework. The specifications are then executable as logic programs. This provides a prototype implementation of the partial evaluator.

Moreover, since deductive system proofs are accessible as objects in Elf, many aspects of the partial evaluator correctness proofs (e.g., the correctness of binding-time analysis) can be coded in Elf and mechanically checked.

1 Introduction

Offline partial evaluation consists of two phases: a binding-time analysis phase (where information is gathered about which parts of the source program depend on known or unknown data), and a specialization phase (where constructs depending on known data are reduced away) [3,15]. Recent work specifies the analysis phase using type systems [7] and the specialization phase using operational semantics [14,25,26]. The type system and operational semantics formalisms can be unified if one emphasizes their logical character: a type-based analysis is a logic for deducing program properties, and an operational semantics is a logic for deducing computational steps or input/output behaviour of programs. However, in program specialization systems that use these formalisms, this logical character has neither been emphasized nor exploited.

In this paper, we exploit this logical character and obtain a uniform framework for specifying, prototyping, and mechanically verifying the correctness of program specialization systems. Specifically, we consider offline partial evalua-

* This work is supported by the Danish Research Academy and by the DART project (Design, Analysis and Reasoning about Tools) of the Danish Research Councils.

** Universitetsparken 1, 2100 Copenhagen Ø, Denmark. E-mail: hatcliff@diku.dk
tion in the style of the partial evaluator $\lambda$-mix [7]. $\lambda$-mix is a good illustrative case since it is simple, and one of the few partial evaluators with a rigorous semantic foundation. It has also spawned additional work on the correctness of binding-time analysis [19,28] and specialization [16].

Our results are as follows.

- We give novel specifications of binding-time constraints and specialization as natural-deduction style logics. These specifications simplify meta-theory activities such as proving the correctness of binding-time analysis and specialization.

- We formalize the specifications using LF — a meta-language (a dependently-typed $\lambda$-calculus) for defining logics [12]. In LF, judgements (assertions) are represented as types, and deductions are represented as objects. Determining the validity of a deduction is reduced to checking if the representing object is well-typed. Since LF type-checking is decidable, purported deductions can be checked automatically for validity.

- We obtain prototypes directly from the formal specifications using Elf — a logic programming language based on LF [20]. Elf gives an operational interpretation to LF types by treating them as goals. Thus, the LF specifications of the binding-time analysis and specializer are directly executable in Elf.

- We formalize and mechanically verify much of the meta-theory of offline partial evaluation (e.g., correctness of binding-time analysis and soundness of the specializer) via meta-programming in Elf. Correctness conditions are formalized as judgements about "lower-level" deductions describing object-language evaluation and transformation. Proofs of correctness are formalized as deductions of the correctness judgements. Elf type-checking mechanically verifies that these deductions (and hence the correctness proofs) are valid.

This methodology of specification/implementation/verification using LF and Elf has been successfully applied in other problem areas [11,17]. In particular, we build on Hannan and Pfenning's work on compiler verification in Elf [11]. They conjectured that their techniques could also be applied to partial evaluation [11, p. 416]. They also identify the verification of transformations based on flow analyses as a "challenging problem, yet to be addressed" [11, p. 415]. The present work addresses both of these points. We confirm their conjecture that LF and Elf can be used for specification/implementation/verification of partial evaluators. Moreover, we give one instance where transformations based on flow analyses can be verified — namely, the specialization of programs based on binding-time analysis.

The rest of the paper is organized as follows. Section 2 summarizes LF and Elf. Section 3 presents the object language and its encoding in Elf. Section 4 presents the specifications of the binding-time analysis and specializer. Section

$^3$ Our setting differs from that of $\lambda$-mix in two ways: (1) we are not concerned with self-applying the partial evaluator, and (2) our object language is typed while $\lambda$-mix's is untyped. However, the techniques here apply equally well to the untyped object language of $\lambda$-mix (in fact, they are simpler in the untyped case).